コンパイラ実験第 2 回

05-191022 平田 賢吾

2019 年 10 月 17 日

問 1

以下にこれまでに課題で提出した, 自由変数を持つ局所関数を含む関数を一つ選んで添付した. これは FL 実験の第 8 回に 問 7 の解答として提出した, OCaml の型制約を解くためのプログラムの compose 関数の一部である.

```
List.fold_left
(fun zs (a,_) ->
    List.filter (fun (b,_) -> a != b) zs)
xs
ys
```

2 行目の局所関数 fun (b, _) → a != b は局所変数 a を含んでいる. まずこれを min-caml 風に書くと次のようになる.

```
let rec f zs (a,_) =
  let rec g (b,_) = a != b
  in
  List.filter g zs
in
List.fold_left f xs ys
```

この関数を自由変数がなくなるように書き換えると下のようになる.

```
type record = {a : int};;

let rec g {a=x} (b,_) = x != b in
let rec f zs (x,_) = List.filter (g {a=x}) zs in
List.fold_left f xs ys;;
```

ちなみにこの関数 f は連想リスト xs, ys に対し, xs から ys にラベルが同じものがあれば取り除く関数である.

考察

局所関数定義を使うと引数を減らすことができて, 可読性が上がった. また, 外部で使うことのない関数をわざわざ外に出さずに, 関数の中にネストしておけば, コードを読むときにそれが何のために作られた関数なの

かはっきりしていい. 外にある関数が多すぎると, どれが何をしているのか直感的にわからない. 特に fun で定義した関数を使うと, 一箇所でしか使わない関数にわざわざ名前をつけなくて済む.

問 3

MinCaml コンパイラの擬似コードに従って, 手でクロージャ変換を行った.

(1)

```
let z = 4 in let rec f x = x ^^e2^^80^^93 z in f 8
```

f : クロージャは作られる. 何故ならば, 自由変数 z が f の内部に存在するからである. ラベルは L_f, 自由変数は z のみ. ただし, z をインライン化すれば作られなくなる.

(2)

```
let rec g x = x ^^e2^^80^^93 2 in g 6
```

g : クロージャは作られない. 何故ならば, 自由変数が含まれない上に, 末尾の g 6 は g の直接的な適用なので直接呼び出せばいいからである.

(3)

```
let rec f x = x ^^e2^^80^^93 1 in f
```

f : クロージャは作られる. 自由変数が含まれない一方で, f が値として参照される可能性があるのでクロージャを作らねばならない. ラベルは L_f, 自由変数はない.

(4)

```
let rec g h = let rec i x = h x in i in g
```

i : クロージャは作られる. 何故ならば, i の定義内に自由変数 h が出現しているからである. ラベルは L_g, 自由変数は h.

g : クロージャは作られる. 何故ならば, 末尾に g があり, g が値として他の関数から参照される可能性があるため, クロージャにしておかねばならないからである. ラベルは L_g, 自由変数はない.

(5)

```
let rec i x = x in
let z = 4 in
let rec f x = i (z ^^e2^^80^^93 5) in
if z < 6 then (i, f 7) else (f, 8)
```

f : クロージャは作られる. 何故ならば, 自由変数 z を含むからである. i は自由変数にはならない. ラベルは L_f, 自由変数は z になる. ただし, 定数畳み込みを行うならばクロージャは作られない (z ¡ 6 が true に変わり, 末尾の (f, 8) も不要になる).

i : クロージャは作られる. 何故ならば返り値に i が参照されうる形で存在するので, クロージャにしておかねばならないからである.

(6)

```
let rec fact x =
  if x = 1 then 1
  else x * fact (x ^^e2^^80^^93 1) in
  fact 6
```

fact : クロージャは作られない. 何故ならば, fact の定義部分で fact は直接適用される形 (擬似コードでの apply_direct) になっていて, さらに x 以外の変数も含まれず, 返り値も fact が外部から参照され得ない形になっているからである.

考察

賢いクロージャ変換の手法は賢いとはいえ, 案外作らなくて済むクロージャは少ないと思った. 特に関数を定義したいときは返り値にその関数を他の関数から参照されうる形で現れることがほとんどのように思える.

しかし, この課題でクロージャ変換した関数は, クロージャ変換の前に実際はもっと最適化が行われることを考えると, 例えば (5) は定数畳み込みで i のクロージャを作らなくて済むようになることを考えると, 賢い手法は他の最適化と合わさって強さを発揮するのかもしれないと考えられる. 例えばインライン化は自由変数を消すことができるため, 相性が良さそうに思われる.